

UNIT11: GEOMETRIC PRIMITIVE

【学習要項】

- ☐ Index buffers
- ☐ Constant buffers
- ☐ World, View and Projection transformation matrices
- ☐ Scale, Rotate and Translate transformation matrices

【演習手順】

1. 幾何プリミティブ（今回は正六面体）を生成・描画するクラスを作成する
2. シェーダーファイルの追加

①HLSL ヘッダーファイル (geometric_primitive.hlsli)

```
1: struct VS_OUT
2: {
3:     float4 position : SV_POSITION;
4:     float4 color : COLOR;
5: };
6: cbuffer OBJECT_CONSTANT_BUFFER : register(b0)
7: {
8:     row_major float4x4 world;
9:     float4 material_color;
10: };
11: cbuffer SCENE_CONSTANT_BUFFER : register(b1)
12: {
13:     row_major float4x4 view_projection;
14:     float4 light_direction;
15: };
```

②頂点シェーダーファイル (geometric_primitive_vs.hlsl)

```
1: #include "geometric_primitive.hlsli"
2: VS_OUT main(float4 position : POSITION, float4 normal : NORMAL)
3: {
4:     VS_OUT vout;
5:     vout.position = mul(position, mul(world, view_projection));
6:
7:     normal.w = 0;
8:     float4 N = normalize(mul(normal, world));
9:     float4 L = normalize(-light_direction);
10:
11:     vout.color.rgb = material_color.rgb * max(0, dot(L, N));
12:     vout.color.a = material_color.a;
13:     return vout;
14: }
```

③ピクセルシェーダーファイル (geometric_primitive_ps.hlsl)

```
1: #include "geometric_primitive.hlsli"
2: float4 main(VS_OUT pin) : SV_TARGET
3: {
4:     return pin.color;
5: }
```

3. geometric_primitive クラスの定義

※必要に応じてヘッダファイルをインクルードする

```
1: class geometric_primitive
2: {
3: public:
4:     struct vertex
5:     {
6:         DirectX::XMFLOAT3 position;
7:         DirectX::XMFLOAT3 normal;
8:     };
9:     struct constants
10:    {
```

UNIT11: GEOMETRIC PRIMITIVE

```
11:     DirectX::XMFLOAT4X4 world;
12:     DirectX::XMFLOAT4 material_color;
13: };
14:
15: private:
16:     Microsoft::WRL::ComPtr<ID3D11Buffer> vertex_buffer;
17:     Microsoft::WRL::ComPtr<ID3D11Buffer> index_buffer;
18:
19:     Microsoft::WRL::ComPtr<ID3D11VertexShader> vertex_shader;
20:     Microsoft::WRL::ComPtr<ID3D11PixelShader> pixel_shader;
21:     Microsoft::WRL::ComPtr<ID3D11InputLayout> input_layout;
22:     Microsoft::WRL::ComPtr<ID3D11Buffer> constant_buffer;
23:
24: public:
25:     geometric_primitive(ID3D11Device* device);
26:     virtual ~geometric_primitive() = default;
27:
28:     void render(ID3D11DeviceContext* immediate_context,
29:         const DirectX::XMFLOAT4X4& world, const DirectX::XMFLOAT4& material_color);
30:
31: protected:
32:     void create_com_buffers(ID3D11Device* device, vertex* vertices, size_t vertex_count,
33:         uint32_t* indices, size_t index_count);
34: };
```

4. geometric_primitive クラスのコンストラクタの実装

```
1: geometric_primitive::geometric_primitive(ID3D11Device* device)
2: {
3:     vertex vertices[24]{};
4:     // サイズが 1.0 の正立方体データを作成する（重心を原点にする）。正立方体のコントロールポイント数は 8 個、
5:     // 1 つのコントロールポイントの位置には法線の向きが異なる頂点が 3 個あるので頂点情報の総数は 8x3=24 個、
6:     // 頂点情報配列（vertices）にすべて頂点の位置・法線情報を格納する。
7:
8:     uint32_t indices[36]{};
9:     // 正立方体は 6 面持ち、1 つの面は 2 つの 3 角形ポリゴンで構成されるので 3 角形ポリゴンの総数は 6x2=12 個、
10:    // 正立方体を描画するために 12 回の 3 角形ポリゴン描画が必要、よって参照される頂点情報は 12x3=36 回、
11:    // 3 角形ポリゴンが参照する頂点情報のインデックス（頂点番号）を描画順に配列（indices）に格納する。
12:    // 時計回りが表面になるように格納すること。
13:
14:    create_com_buffers(device, vertices, 24, indices, 36);
15:
16:    HRESULT hr{ S_OK };
17:
18:    D3D11_INPUT_ELEMENT_DESC input_element_desc[]
19:    {
20:        { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0,
21:            D3D11_APPEND_ALIGNED_ELEMENT, D3D11_INPUT_PER_VERTEX_DATA, 0 },
22:        { "NORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0,
23:            D3D11_APPEND_ALIGNED_ELEMENT, D3D11_INPUT_PER_VERTEX_DATA, 0 },
24:    };
25:    create_vs_from_cso(device, "geometric_primitive_vs.cso", vertex_shader.GetAddressOf(),
26:        input_layout.GetAddressOf(), input_element_desc, ARRAYSIZE(input_element_desc));
27:    create_ps_from_cso(device, "geometric_primitive_ps.cso", pixel_shader.GetAddressOf());
28:
29:    D3D11_BUFFER_DESC buffer_desc{};
30:    buffer_desc.ByteWidth = sizeof(constants);
31:    buffer_desc.Usage = D3D11_USAGE_DEFAULT;
32:    buffer_desc.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
33:    hr = device->CreateBuffer(&buffer_desc, nullptr, constant_buffer.GetAddressOf());
34:    _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
35: }
```

5. geometric_primitive クラスの create_com_buffers メンバ関数の実装

UNIT11: GEOMETRIC PRIMITIVE

```
1: void geometric_primitive::create_com_buffers(ID3D11Device* device, vertex* vertices, size_t vertex_count,
2:   uint32_t* indices, size_t index_count)
3: {
4:   HRESULT hr{ S_OK };
5:
6:   D3D11_BUFFER_DESC buffer_desc{};
7:   D3D11_SUBRESOURCE_DATA subresource_data{};
8:   buffer_desc.ByteWidth = static_cast<UINT>(sizeof(vertex) * vertex_count);
9:   buffer_desc.Usage = D3D11_USAGE_DEFAULT;
10:  buffer_desc.BindFlags = D3D11_BIND_VERTEX_BUFFER;
11:  buffer_desc.CPUAccessFlags = 0;
12:  buffer_desc.MiscFlags = 0;
13:  buffer_desc.StructureByteStride = 0;
14:  subresource_data.pSysMem = vertices;
15:  subresource_data.SysMemPitch = 0;
16:  subresource_data.SysMemSlicePitch = 0;
17:  hr = device->CreateBuffer(&buffer_desc, &subresource_data, vertex_buffer.ReleaseAndGetAddressOf());
18:  _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
19:
20:  buffer_desc.ByteWidth = static_cast<UINT>(sizeof(uint32_t) * index_count);
21:  buffer_desc.Usage = D3D11_USAGE_DEFAULT;
22:  buffer_desc.BindFlags = D3D11_BIND_INDEX_BUFFER;
23:  subresource_data.pSysMem = indices;
24:  hr = device->CreateBuffer(&buffer_desc, &subresource_data, index_buffer.ReleaseAndGetAddressOf());
25:  _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
26: }
```

6. geometric_primitive クラスの render メンバ関数の実装

```
1: void geometric_primitive::render(ID3D11DeviceContext* immediate_context,
2:   const DirectX::XMFLOAT4X4& world, const DirectX::XMFLOAT4& material_color)
3: {
4:   uint32_t stride{ sizeof(vertex) };
5:   uint32_t offset{ 0 };
6:   immediate_context->IASetVertexBuffers(0, 1, vertex_buffer.GetAddressOf(), &stride, &offset);
7:   immediate_context->IASetIndexBuffer(index_buffer.Get(), DXGI_FORMAT_R32_UINT, 0);
8:   immediate_context->IASetPrimitiveTopology(D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
9:   immediate_context->IASetInputLayout(input_layout.Get());
10:
11:  immediate_context->VSSetShader(vertex_shader.Get(), nullptr, 0);
12:  immediate_context->PSSetShader(pixel_shader.Get(), nullptr, 0);
13:
14:  constants data{ world, material_color };
15:  immediate_context->UpdateSubresource(constant_buffer.Get(), 0, 0, &data, 0, 0);
16:  immediate_context->VSSetConstantBuffers(0, 1, constant_buffer.GetAddressOf());
17:
18:  D3D11_BUFFER_DESC buffer_desc{};
19:  index_buffer->GetDesc(&buffer_desc);
20:  immediate_context->DrawIndexed(buffer_desc.ByteWidth / sizeof(uint32_t), 0, 0);
21: }
```

7. シーン定数バッファ

①framework クラスに定義する

```
1: struct scene_constants
2: {
3:     DirectX::XMFLOAT4X4 view_projection;           //ビュー・プロジェクション変換行列
4:     DirectX::XMFLOAT4 light_direction;             //ライトの向き
5: };
6: Microsoft::WRL::ComPtr<ID3D11Buffer> constant_buffers[8];
```

②framework クラスのコンストラクタでシーン定数バッファオブジェクトを生成する

```
1: D3D11_BUFFER_DESC buffer_desc{};
2: buffer_desc.ByteWidth = sizeof(scene_constants);
```

UNIT11: GEOMETRIC PRIMITIVE

```
3: buffer_desc.Usage = D3D11_USAGE_DEFAULT;
4: buffer_desc.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
5: buffer_desc.CPUAccessFlags = 0;
6: buffer_desc.MiscFlags = 0;
7: buffer_desc.StructureByteStride = 0;
8: hr = device->CreateBuffer(&buffer_desc, nullptr, constant_buffers[0].GetAddressOf());
9: _ASSERT_EXPR(SUCCEEDED(hr), hr_trace(hr));
```

- ③framework クラスの render メンバ変数でビュー・プロジェクション変換行列を計算し、それを定数バッファにセットする

※ネームスペース (DirectX) は省略してある

```
1: D3D11_VIEWPORT viewport;
2: UINT num_viewports{ 1 };
3: immediate_context->RSGetViewports(&num_viewports, &viewport);
4:
5: float aspect_ratio{ viewport.Width / viewport.Height };
6: XMATRIX P{ XMMatrixPerspectiveFovLH(XMConvertToRadians(30), aspect_ratio, 0.1f, 100.0f) };
7:
8: XMVECTOR eye{ XMVectorSet(0.0f, 0.0f, -10.0f, 1.0f) };
9: XMVECTOR focus{ XMVectorSet(0.0f, 0.0f, 0.0f, 1.0f) };
10: XMVECTOR up{ XMVectorSet(0.0f, 1.0f, 0.0f, 0.0f) };
11: XMATRIX V{ XMMatrixLookAtLH(eye, focus, up) };
12:
13: scene_constants data{};
14: XMStoreFloat4x4(&data.view_projection, V * P);
15: data.light_direction = { 0, 0, 1, 0 };
16: immediate_context->UpdateSubresource(constant_buffers[0].Get(), 0, 0, &data, 0, 0);
17: immediate_context->VSSetConstantBuffers(1, 1, constant_buffers[0].GetAddressOf());
```

8. framework クラスのメンバ変数として geometric_primitive*型配列を要素数 8 で宣言する

```
std::unique_ptr<geometric_primitive> geometric_primitives[8];
```

9. framework クラスの initialize メンバ関数で geometric_primitive オブジェクトを生成する

```
geometric_primitives[0] = std::make_unique<geometric_primitive>(device.Get());
```

10. framework クラスの render メンバ関数で geometric_primitive クラスの render メンバ関数を呼び出す

- ①拡大縮小 (S)・回転 (R)・平行移動 (T) 行列を計算する

```
DirectX::XMATRIX S{ DirectX::XMMatrixScaling(1, 1, 1) };
DirectX::XMATRIX R{ DirectX::XMMatrixRotationRollPitchYaw(0, 0, 0) };
DirectX::XMATRIX T{ DirectX::XMMatrixTranslation(0, 0, 0) };
```

- ②上記 3 行列を合成しワールド変換行列を作成する

```
DirectX::XMFLOAT4X4 world;
DirectX::XMStoreFloat4x4(&world, S * R * T);
```

- ③geometric_primitive クラスの render メンバ関数を呼び出す

※深度テスト：オン、深度ライト：オンの深度ステンシルステートをバインドしておくこと

```
geometric_primitives[0]->render(immediate_context.Get(), world, { 0.5f, 0.8f, 0.2f, 1.0f });
```

11. 実行し、正立方体が描画される事を確認する
12. カメラの位置・ライトの照射方向を ImGui を使って実行時に変更できるようにする
13. 正立方体の位置・姿勢・寸法・色を ImGui を使って実行時に変更できるようにする

【評価項目】

- ☐ 正立方体の描画
☐ 正立方体の位置・姿勢・寸法・色の変更